ClickHouse

第一章 ClickHouse简介及特点

简介

ClickHouse是俄罗斯的Yandex于2016 年开源的`列式存储数据库（DBMS）`，使用 C++ 语言编写，主要用于`在线分析处理查询（OLAP）`，能够使用SQL查询实时生成分析数据报告。

特点

列式存储

行式存储：好处是想查某个人所有的属性时，可以通过一次磁盘查找加顺序读取就可以。但是当想查所有人的年龄时，需要不停的查找，或者全表扫描才行，遍历的很多数据都是不需要的。

列式存储：想查所有人的年龄只需把年龄那一列拿出来即可。

列式存储的好处：

1.对于列的聚合，计数，求和等统计操作优于行式存储。

2.由于某一列的数据类型都是相同的，针对于数据存储更容易进行数据压缩，每一列选择更优的数据压缩算法，大大提高了数据的压缩比重。

3.由于数据压缩比更好，一方面节省了磁盘空间，另一方面对于cache也有了更大的发挥空间。

DBMS的功能

几乎覆盖了标准 SQL 的大部分语法，包括 DDL 和 DML，以及配套的各种函数，用户管理及权限管理，数据的备份与恢复。

多样化引擎

ClickHouse 和 MySQL 类似，把表级的存储引擎插件化，根据表的不同需求可以设定不同 的存储引擎。目前包括合并树、日志、接口和其他四大类 20 多种引擎。

高吞吐写入能力

ClickHouse 采用类 LSM Tree的结构，数据写入后定期在后台 Compaction。通过类 LSM tree 的结构，ClickHouse 在数据导入时全部是顺序 append 写，写入后数据段不可更改，在后台 compaction 时也是多个段 merge sort 后顺序写回磁盘。顺序写的特性，充分利用了磁盘的吞 吐能力，即便在 HDD 上也有着优异的写入性能。

官方公开 benchmark 测试显示能够达到 50MB-200MB/s 的写入吞吐能力，按照每行 100Byte 估算，大约相当于 50W-200W 条/s 的写入速度。

数据分区与线程级并行

ClickHouse 将数据划分为多个 partition，每个 partition 再进一步划分为多个 index granularity(索引粒度)，然后通过多个 CPU核心分别处理其中的一部分来实现并行数据处理。 在这种设计下，单条 Query 就能利用整机所有 CPU。极致的并行处理能力，极大的降低了查 询延时。

所以，ClickHouse 即使对于大量数据的查询也能够化整为零平行处理。但是有一个弊端 就是对于单条查询使用多 cpu，就不利于同时并发多条查询。所以对于高 qps 的查询业务， ClickHouse 并不是强项。

性能对比

ClickHouse 像很多 OLAP 数据库一样，单表查询速度由于关联查询，而且 ClickHouse 的两者差距更为明显。

第二章 ClickHouse表引擎

表引擎是 ClickHouse 的一大特色。可以说， 表引擎决定了如何存储表的数据。包括：

1.数据的存储方式和位置，写到哪里以及从哪里读取数据。

2.支持哪些查询以及如何支持。

3.并发数据访问。

4.索引的使用（如果存在。

5.是否可以执行多线程请求。

6.数据复制参数。 表引擎的使用方式就是必须显式在创建表时定义该表使用的引擎，以及引擎使用的相关 参数。

特别注意：引擎的名称大小写敏感

1 MaterializeMySQL

ClickHouse 20.8.2.3 版本新增加了MaterializeMySQL的database引擎，该database能映射到MySQL中的某个database，并自动在ClickHouse中创建对应的ReplacingMergeTree。

ClickHouse服务做为MySQL副本，读取Binlog 并执行DDL和DML请求，实现了基于MySQL Binlog机制的业务数据库实时同步功能。

```
打开/etc/my.cnf,在[mysqld]下添加：

server-id=1
log-bin=mysql-bin
binlog_format=ROW

gtid-mode=on
enforce-gtid-consistency=1 # 设置为主从强一致性
log-slave-updates=1 # 记录日志
```

GTID是MySQL复制增强版，从MySQL5.6版本开始支持，目前已经是MySQL主流复制模式。它为每个event分配一个全局唯一ID和序号，我们可以不用关心MySQL集群主从拓扑结构，直接告知MySQL这个GTID即可。

```
-- 在mysql中创建表并写入数据
CREATE DATABASE testck;

CREATE TABLE `testck`.`t_organization` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
```

```
  `code` int NOT NULL,
  `name` text DEFAULT NULL,
  `updatetime` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY (`code`)
 ) ENGINE=InnoDB;

 INSERT INTO testck.t_organization (code, name,updatetime) VALUES(1000,'Realinsight',NOW());
 INSERT INTO testck.t_organization (code, name,updatetime) VALUES(1001, 'Realindex',NOW());
 INSERT INTO testck.t_organization (code, name,updatetime) VALUES(1002,'EDT',NOW());


 CREATE TABLE `testck`.`t_user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `code` int,
  PRIMARY KEY (`id`)
 ) ENGINE=InnoDB;

 INSERT INTO testck.t_user (code) VALUES(1);
```

```
-- 开启ClickHouse物化引擎（必须开）
set allow_experimental_database_materialize_mysql=1;
```

```
-- 在clickhouse中创建数据库
CREATE DATABASE test_binlog ENGINE = MaterializeMySQL('master:3306','testck','root','123456');
```

```
-- 查看clickhouse的数据
use test_binlog;
show tables;
select * from t_organization;
select * from t_user;
```

2 MergeTree

ClickHouse 中最强大的表引擎当属 MergeTree（合并树）引擎及该系列（*MergeTree）中的其他引擎，支持索引和分区，地位可以相当于 innodb 之于 Mysql。而且基于 MergeTree， 还衍生除了很多小弟，也是非常有特色的引擎。

```
create table t_order_mt(
 id UInt32,
 sku_id String,
 total_amount Decimal(16,2),
 create_time Datetime
) engine =MergeTree
  partition by toYYYYMMDD(create_time)
  primary key (id)
  order by (id,sku_id);
```

partition by（可选参数）

分区的目的主要是降低扫描的范围，优化查询速度。

如果不填，就会使用一个分区。

分区目录：MergeTree 是以列文件+索引文件+表定义文件组成的，但是如果设定了分区那么这些文 件就会保存到不同的分区目录中。

并行：分区后，面对涉及跨分区的查询统计，ClickHouse 会以分区为单位并行处理。

数据写入与分区合并：任何一个批次的数据写入都会产生一个临时分区，不会纳入任何一个已有的分区。写入 后的某个时刻（大概 10-15 分钟后），ClickHouse会自动执行合并操作（等不及也可以手动通过 optimize 执行），把临时分区的数据，合并到已有分区中。

```
optimize table xxxx final;
```

primary key（可选参数）

ClickHouse 中的主键，和其他数据库不太一样，它只提供了数据的一级索引，但是却不是唯一约束。这就意味着是可以存在相同 primary key 的数据的。

主键的设定主要依据是查询语句中的 where 条件。根据条件通过对主键进行某种形式的二分查找，能够定位到对应的 index granularity,避免了全表扫描。

index granularity： 直接翻译的话就是索引粒度，指在稀疏索引中两个相邻索引对应数据的间隔。ClickHouse 中的 MergeTree 默认是 8192。官方不建议修改这个值，除非该列存在 大量重复值，比如在一个分区中几万行才有一个不同数据。

稀疏索引：稀疏索引的好处就是可以用很少的索引数据，定位更多的数据，代价就是只能定位到索 引粒度的第一行，然后再进行进行一点扫描。

order by（必选）

order by 设定了分区内的数据按照哪些字段顺序进行有序保存。

order by 是 MergeTree 中唯一一个必填项，甚至比 primary key 还重要，因为当用户不 设置主键的情况，很多处理会依照 order by 的字段进行处理（比如后面会讲的去重和汇总）。

要求：主键必须是 order by 字段的前缀字段。

比如 order by 字段是 (id,sku_id) 那么主键必须是 id 或者(id,sku_id)

二级索引

目前在 ClickHouse 的官网上二级索引的功能在 v20.1.2.4 之前是被标注为实验性的，在 这个版本之后默认是开启的。

是否允许使用实验性的二级索引（v20.1.2.4 开始，这个参数已被删除，默认开启）

```
set allow_experimental_data_skipping_indices=1;
```

```
create table t_order_mt2(
 id UInt32,
 sku_id String,
 total_amount Decimal(16,2),
 create_time Datetime,
INDEX a total_amount TYPE minmax GRANULARITY 5
) engine =MergeTree
  partition by toYYYYMMDD(create_time)
  primary key (id)
  order by (id, sku_id);
```

其中 GRANULARITY N 是设定二级索引对于一级索引粒度的粒度。

```
insert into t_order_mt2 values
(101,'sku_001',1000.00,'2020-06-01 12:00:00') ,
(102,'sku_002',2000.00,'2020-06-01 11:00:00'),
(102,'sku_004',2500.00,'2020-06-01 12:00:00'),
(102,'sku_002',2000.00,'2020-06-01 13:00:00'),
(102,'sku_002',12000.00,'2020-06-01 13:00:00'),
(102,'sku_002',600.00,'2020-06-02 12:00:00');
```

那么在使用下面语句进行测试，可以看出二级索引能够为非主键字段的查询发挥作用。

```
clickhouse-client --send_logs_level=trace <<< 'select
* from t_order_mt2 where total_amount > toDecimal32(900., 2)';
```

数据TTL

TTL 即 Time To Live，MergeTree 提供了可以管理数据表或者列的生命周期的功能。

```
create table t_order_mt3(
 id UInt32,
 sku_id String,
 total_amount Decimal(16,2) TTL create_time+interval 10 SECOND,
 create_time Datetime
) engine =MergeTree
partition by toYYYYMMDD(create_time)
 primary key (id)
 order by (id, sku_id);
```

```
insert into t_order_mt3 values
(106,'sku_001',1000.00,'2020-06-12 22:52:30'),
(107,'sku_002',2000.00,'2020-06-12 22:52:30'),
(110,'sku_003',600.00,'2020-06-13 12:00:00');
```

手动合并，查看效果到期后，指定的字段数据归 0

下面的这条语句是数据会在 create_time 之后 10 秒丢失

```
alter table t_order_mt3 MODIFY TTL create_time + INTERVAL 10 SECOND;
```

涉及判断的字段必须是 Date 或者 Datetime 类型，推荐使用分区的日期字段。

3 ReplacingMergeTree

ReplacingMergeTree是MergeTree 的一个变种，它存储特性完全继承 MergeTree，只是多了一个去重的功能。 尽管 MergeTree 可以设置主键，但是 primary key 其实没有唯一约束的功能。如果你想处理掉重复的数据，可以借助这个 ReplacingMergeTree。

去重时机：数据的去重只会在合并的过程中出现。合并会在未知的时间在后台进行，所以你无法预先作出计划。有一些数据可能仍未被处理。

去重范围：如果表经过了分区，去重只会在分区内部进行去重，不能执行跨分区的去重。 所以 ReplacingMergeTree 能力有限， ReplacingMergeTree 适用于在后台清除重复的数 据以节省空间，但是它不保证没有重复的数据出现。

```
create table t_order_rmt(
 id UInt32,
 sku_id String,
 total_amount Decimal(16,2) ,
 create_time Datetime
) engine =ReplacingMergeTree(create_time)
 partition by toYYYYMMDD(create_time)
 primary key (id)
 order by (id, sku_id);
```

ReplacingMergeTree() 填入的参数为版本字段，重复数据保留版本字段值最大的。

如果不填版本字段，默认按照插入顺序保留最后一条。

```
insert into t_order_rmt values
(101,'sku_001',1000.00,'2020-06-01 12:00:00') ,
(102,'sku_002',2000.00,'2020-06-01 11:00:00'),
(102,'sku_004',2500.00,'2020-06-01 12:00:00'),
(102,'sku_002',2000.00,'2020-06-01 13:00:00'),
(102,'sku_002',12000.00,'2020-06-01 13:00:00'),
(102,'sku_002',600.00,'2020-06-02 12:00:00');
```

执行第一次查询

```
select * from t_order_rmt;
```

手动合并

```
OPTIMIZE TABLE t_order_rmt FINAL;
```

再执行一次查询

```
select * from t_order_rmt;
```

通过测试得到结论：

1.实际上是使用 order by 字段作为唯一键

2.去重不能跨分区

3.只有同一批插入（新版本）或合并分区时才会进行去重

4.认定重复的数据保留，版本字段值最大的

5.如果版本字段相同则按插入顺序保留最后一笔

4 SummingMergeTree

对于不查询明细，只关心以维度进行汇总聚合结果的场景。如果只使用普通的MergeTree 的话，无论是存储空间的开销，还是查询时临时聚合的开销都比较大。

ClickHouse 为了这种场景，提供了一种能够“预聚合”的引擎 SummingMergeTree。

```
create table t_order_smt(
 id UInt32,
 sku_id String,
 total_amount Decimal(16,2) ,
 create_time Datetime
) engine =SummingMergeTree(total_amount)
 partition by toYYYYMMDD(create_time)
 primary key (id)
 order by (id,sku_id );
```

```
insert into t_order_smt values
(101,'sku_001',1000.00,'2020-06-01 12:00:00'),
(102,'sku_002',2000.00,'2020-06-01 11:00:00'),
(102,'sku_004',2500.00,'2020-06-01 12:00:00'),
(102,'sku_002',2000.00,'2020-06-01 13:00:00'),
(102,'sku_002',12000.00,'2020-06-01 13:00:00'),
(102,'sku_002',600.00,'2020-06-02 12:00:00');
```

```
select * from t_order_smt;
```

```
OPTIMIZE TABLE t_order_smt FINAL;
```

```
select * from t_order_smt;
```

通过结果可以得到以下结论：

1.以 SummingMergeTree（）中指定的列作为汇总数据列

2.可以填写多列必须数字列，如果不填，以所有非维度列且为数字列的字段为汇总数据列

3.以 order by 的列为准，作为维度列

4.其他的列按插入顺序保留第一行

5.不在一个分区的数据不会被聚合

6.只有在同一批次插入(新版本)或分片合并时才会进行聚合

开发建议：设计聚合表的话，唯一键值、流水号可以去掉，所有字段全部是维度、度量或者时间戳。

问题：

能不能直接执行以下 SQL 得到汇总值

```
select total_amount from XXX where province_name=’ ’  and create_date=’ xxx’
```

不行，可能会包含一些还没来得及聚合的临时明细

如果要是获取汇总值，还是需要使用 sum 进行聚合，这样效率会有一定的提高，但本 身 ClickHouse 是列式存储的，效率提升有限，不会特别明显。

```
select sum(total_amount) from province_name=’ ’  and create_date= ‘xxx’
```

5 TinyLog和Memory

TinyLog

以列文件的形式保存在磁盘上，不支持索引，没有并发控制。一般保存少量数据的小表， 生产环境上作用有限。可以用于平时练习测试用。

```
1    create table t_tinylog ( id String, name String) engine=TinyLog;
```

Memory

内存引擎，数据以未压缩的原始形式直接保存在内存当中，服务器重启数据就会消失。 读写操作不会相互阻塞，不支持索引。简单查询下有非常非常高的性能表现（超过 10G/s）。

一般用到它的地方不多，除了用来测试，就是在需要非常高的性能，同时数据量又不太 大（上限大概 1 亿行）的场景。